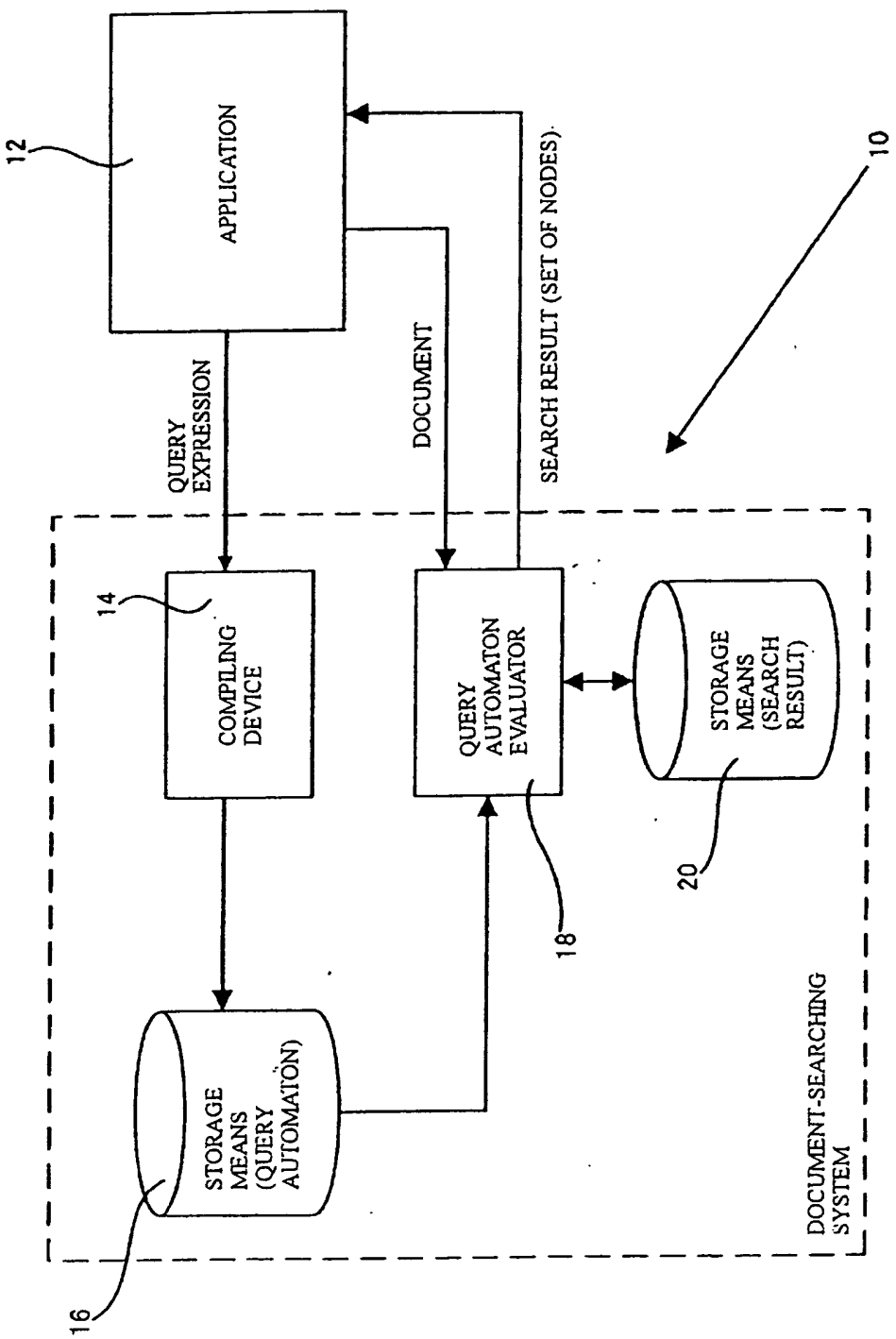

12
APPLICATION
10
QUERY EXPRESSION
DOCUMENT
SEARCH RESULT (SET OF NODES)
14
COMPILING DEVICE
QUERY AUTOMATON EVALUATOR
STORAGE MEANS (SEARCH RESULT)
20
18
STORAGE MEANS (QUERY AUTOMATON)
16
DOCUMENT-SEARCHING SYSTEM

Figure 1

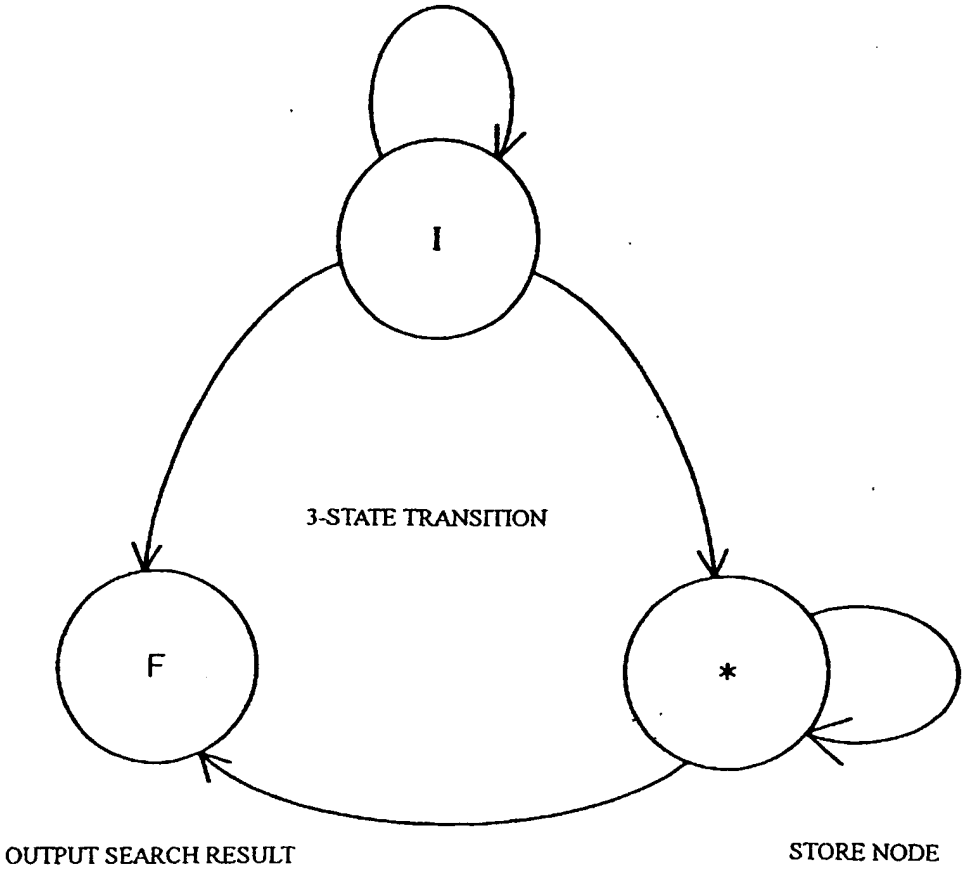


Figure 2

```
<html>
 <head><title>Sample</title></head>
 <body>
   <div>
     <p>A sample document</p>
   </div>
   <p>A sample paragraph in div</p>
 </body>
</html>
```

Figure 3

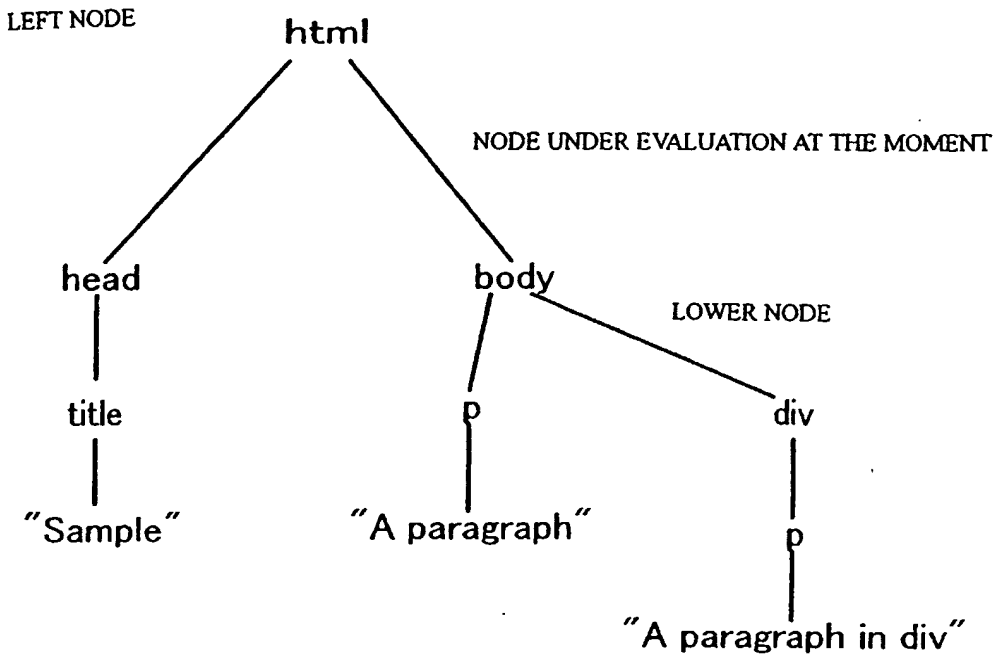


Figure 4

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	I, F	¬p	0, 1	0, 1
1	*, I, F	any	0, 1	0, 1

Figure 5

QUERY AUTOMATON FOR //p[not(ancestor::div)]

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	I, F	¬p	0, 1	0, 1
0	I, F	div	1	2
1	*I, F	any	2	0,1
2	I	any	0,1	2

(a)

QUERY AUTOMATON FOR //*[.//div and.//p]

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	F	¬div	0, 1	2
0	F	¬p	0, 1	3
1	*, I, F	any	0, 1	0, 1
2	I	¬div	2	2
3	I	¬p	3	3

(b)

Figure 6

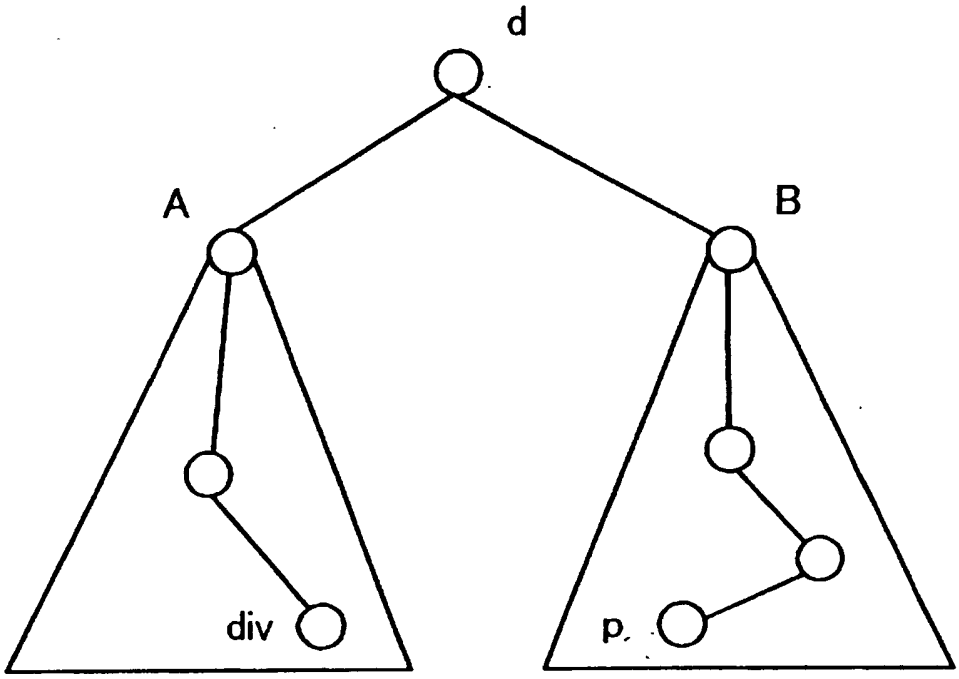


Figure 7

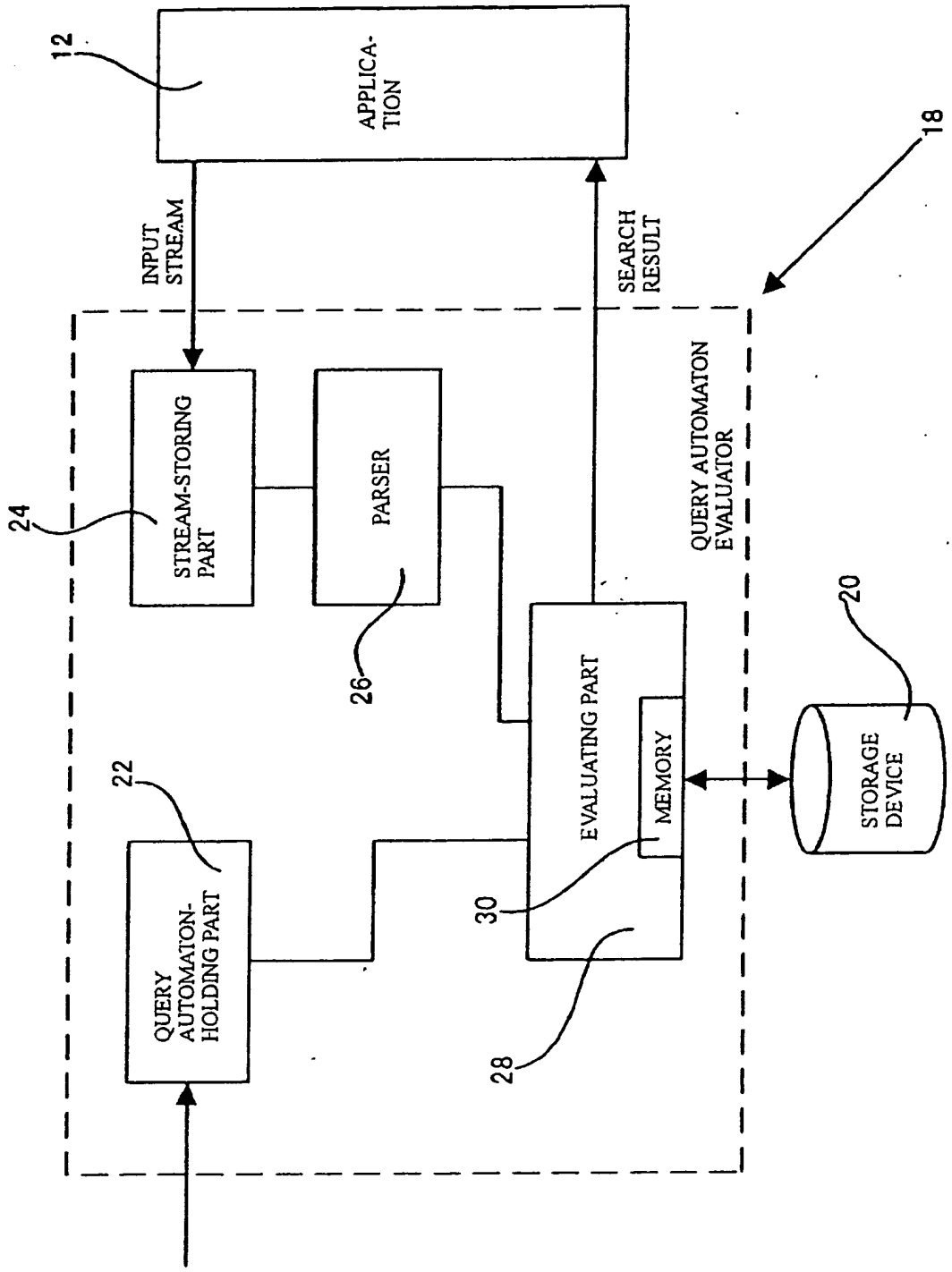

12
APPLICA-
TION
18
INPUT
STREAM
SEARCH
RESULT
QUERY AUTOMATON
EVALUATOR
24
STREAM-STORING
PART
PARSER
26
EVALUATING PART
MEMORY
20
STORAGE
DEVICE
22
QUERY
AUTOMATON-
HOLDING PART
30
28

Figure 8

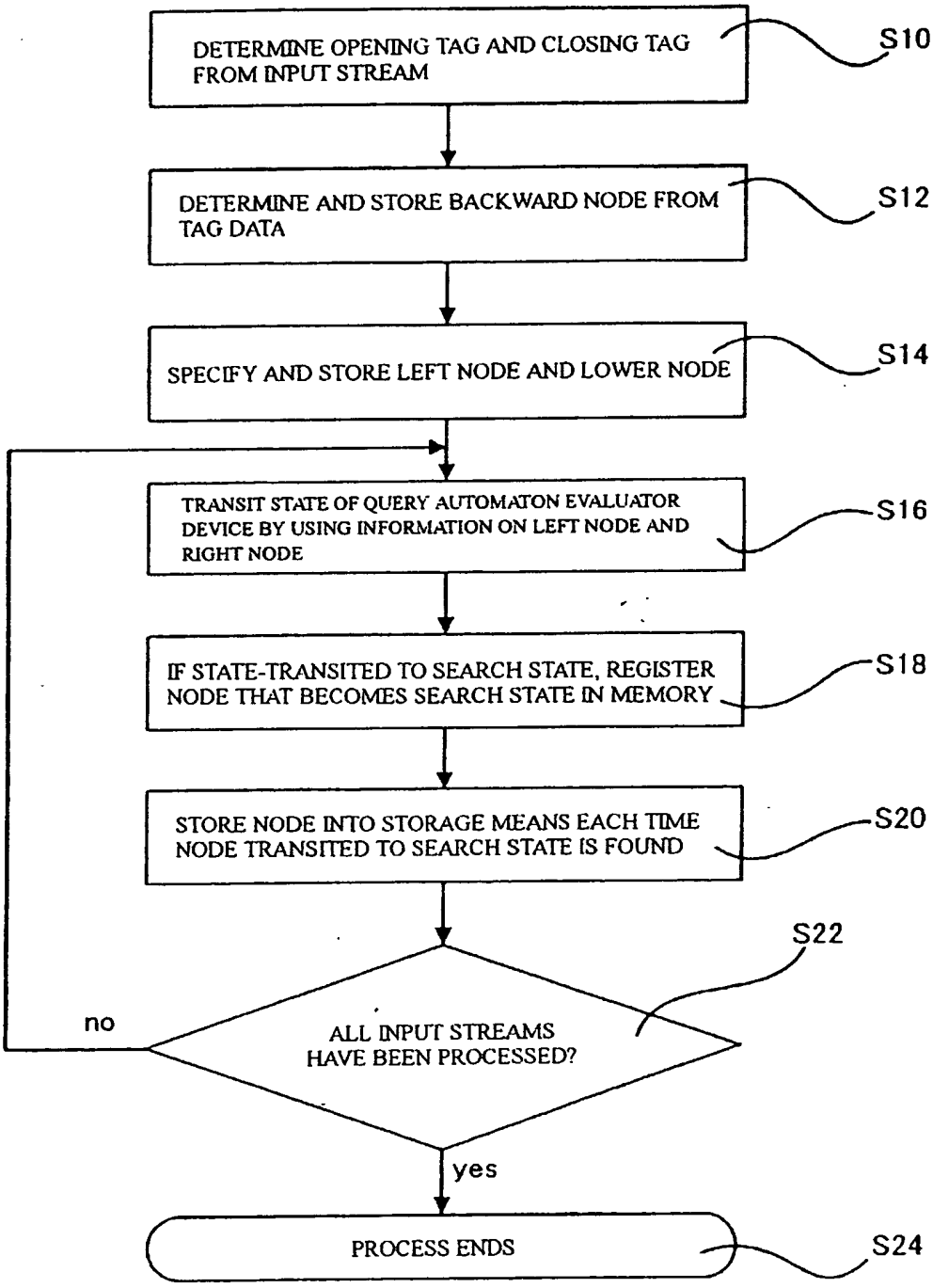

DETERMINE OPENING TAG AND CLOSING TAG FROM INPUT STREAM
S10
DETERMINE AND STORE BACKWARD NODE FROM TAG DATA
S12
SPECIFY AND STORE LEFT NODE AND LOWER NODE
S14
TRANSIT STATE OF QUERY AUTOMATON EVALUATOR DEVICE BY USING INFORMATION ON LEFT NODE AND RIGHT NODE
S16
IF STATE-TRANSITED TO SEARCH STATE, REGISTER NODE THAT BECOMES SEARCH STATE IN MEMORY
S18
STORE NODE INTO STORAGE MEANS EACH TIME NODE TRANSITED TO SEARCH STATE IS FOUND
S20
S22
ALL INPUT STREAMS HAVE BEEN PROCESSED?
no
yes
PROCESS ENDS
S24

Figure 9

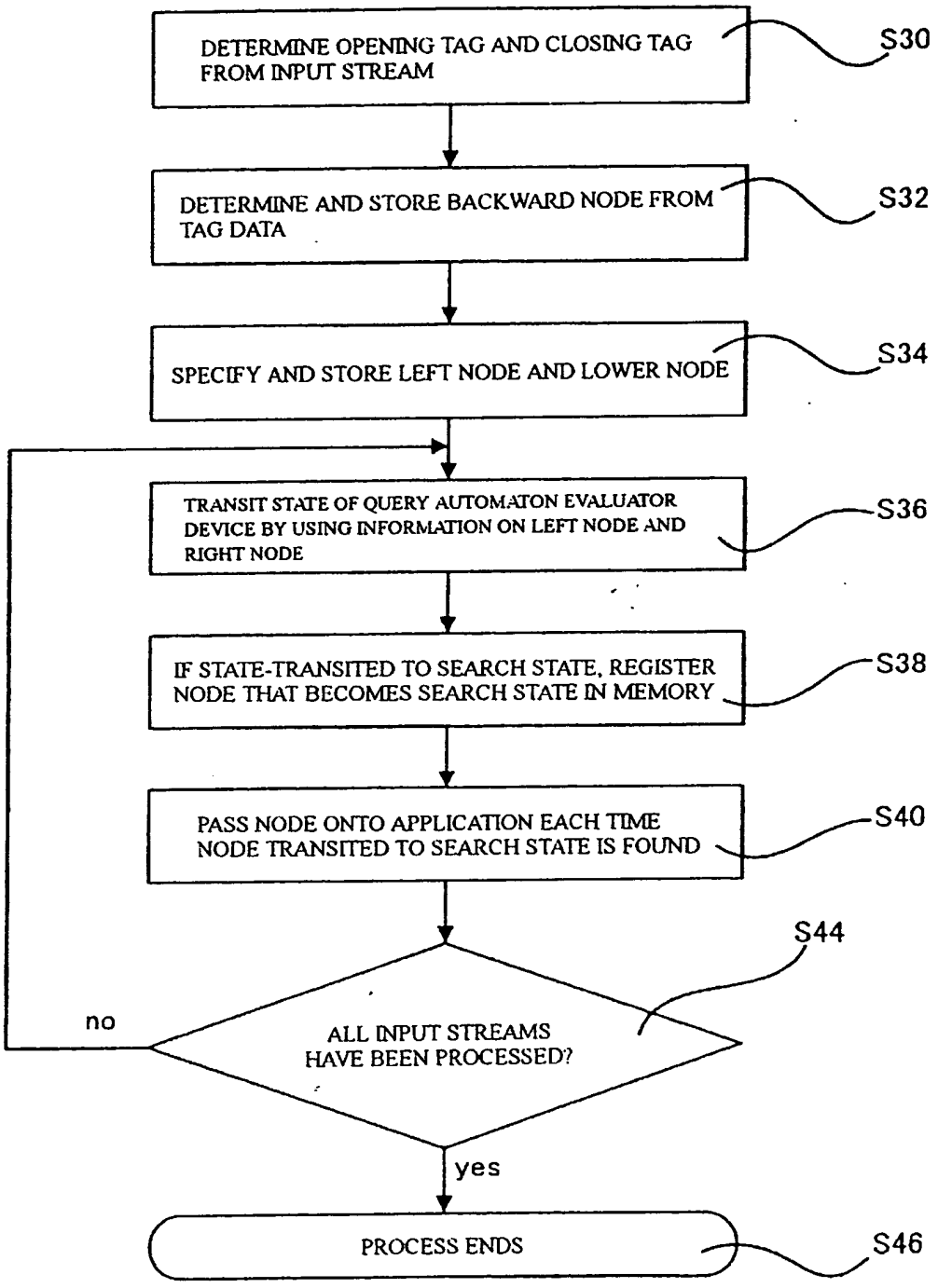

DETERMINE OPENING TAG AND CLOSING TAG FROM INPUT STREAM
S30
DETERMINE AND STORE BACKWARD NODE FROM TAG DATA
S32
SPECIFY AND STORE LEFT NODE AND LOWER NODE
S34
TRANSIT STATE OF QUERY AUTOMATON EVALUATOR DEVICE BY USING INFORMATION ON LEFT NODE AND RIGHT NODE
S36
IF STATE-TRANSITED TO SEARCH STATE, REGISTER NODE THAT BECOMES SEARCH STATE IN MEMORY
S38
PASS NODE ONTO APPLICATION EACH TIME NODE TRANSITED TO SEARCH STATE IS FOUND
S40
S44
ALL INPUT STREAMS HAVE BEEN PROCESSED?
no
yes
PROCESS ENDS
S46

Figure 10

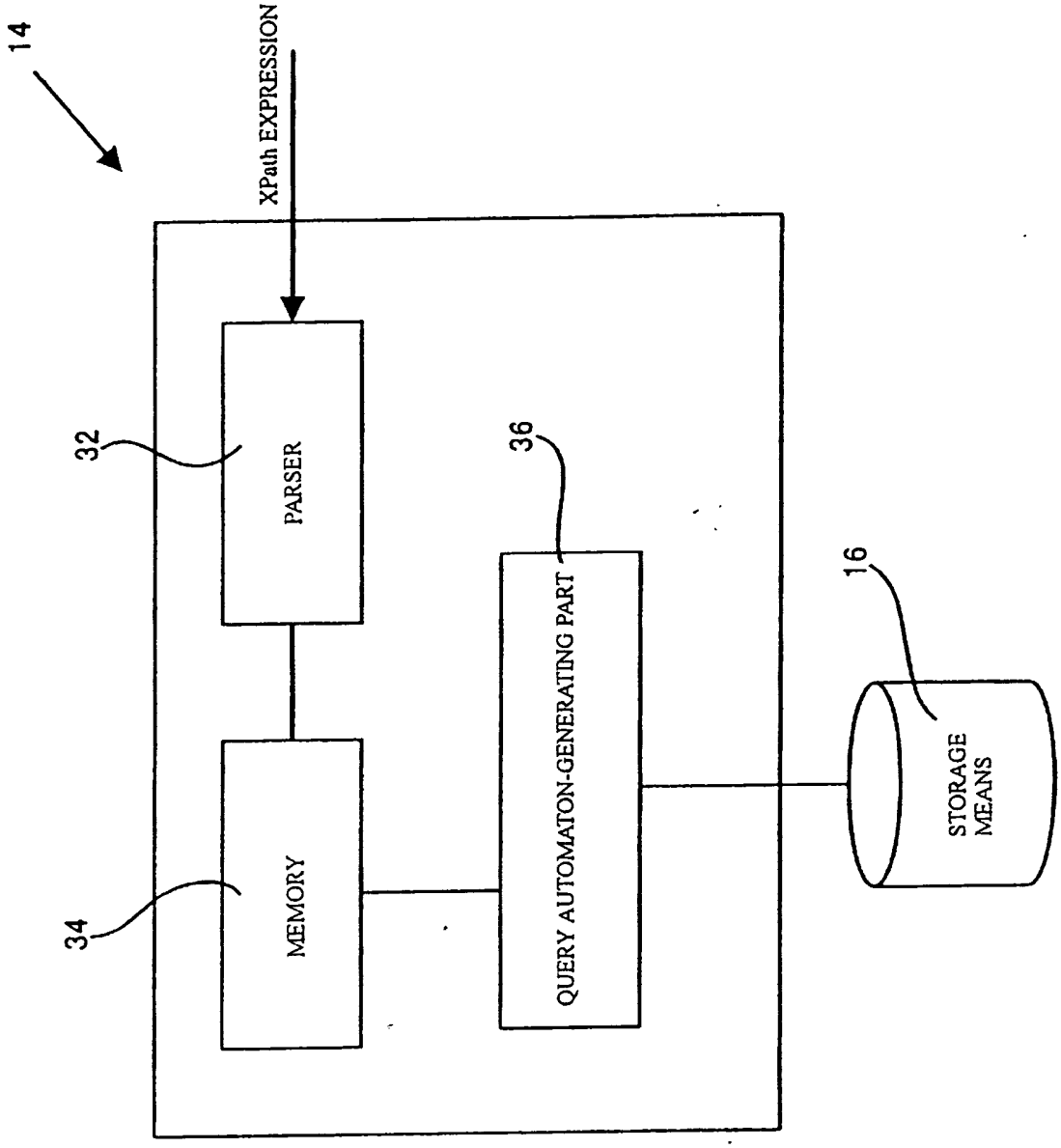

14
XPath EXPRESSION
32
PARSER
36
QUERY AUTOMATON-GENERATING PART
16
STORAGE MEANS
34
MEMORY

Figure 11

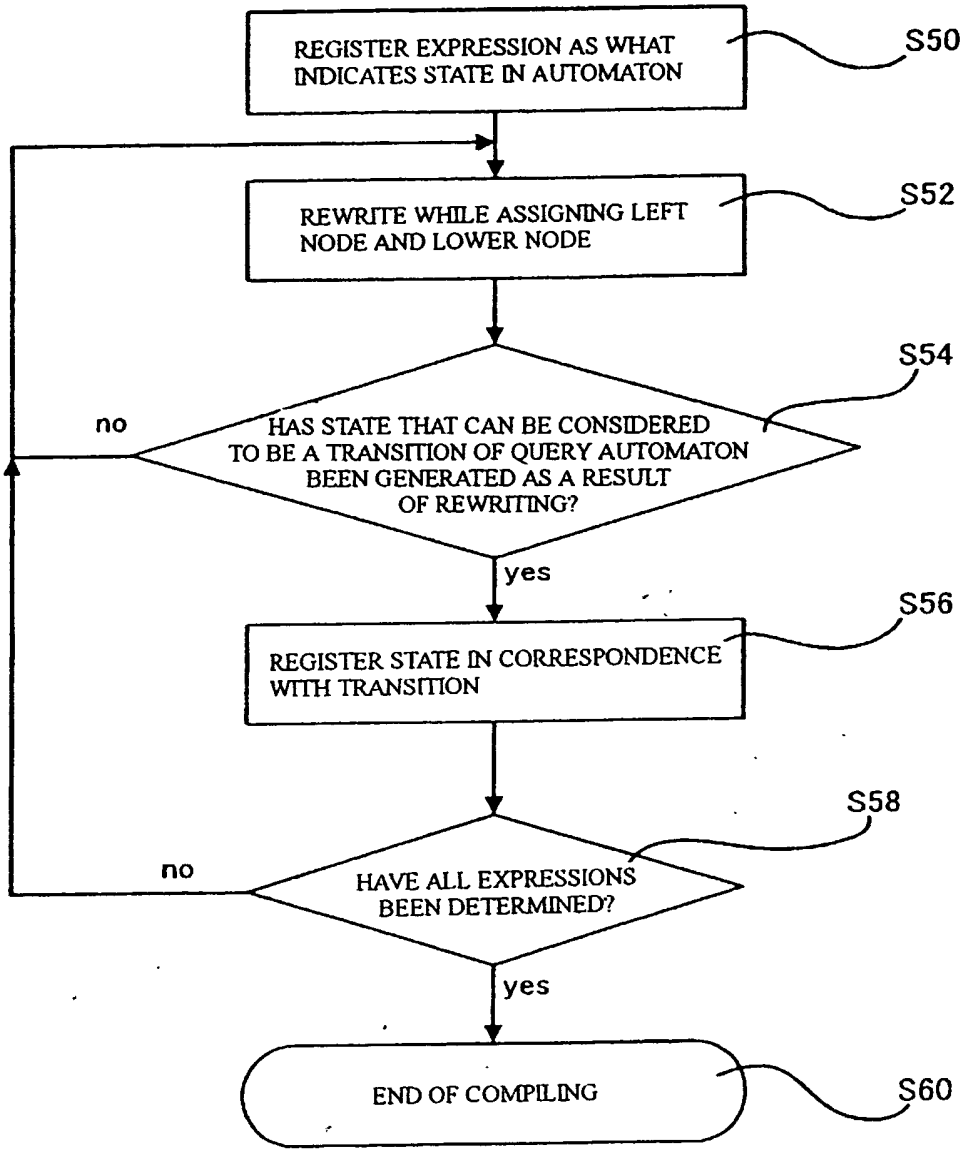

REGISTER EXPRESSION AS WHAT INDICATES STATE IN AUTOMATON
S50
REWRITE WHILE ASSIGNING LEFT NODE AND LOWER NODE
S52
S54
HAS STATE THAT CAN BE CONSIDERED TO BE A TRANSITION OF QUERY AUTOMATON BEEN GENERATED AS A RESULT OF REWRITING?
no
yes
REGISTER STATE IN CORRESPONDENCE WITH TRANSITION
S56
S58
HAVE ALL EXPRESSIONS BEEN DETERMINED?
no
yes
END OF COMPILING
S60

Figure 12

```
1 proc eval(v)
2   case v
3     u ⟨σ⟩ w ⟨/σ⟩ → Ξ := { ⟨q, {x | ∀q', q''. q ∈ δ(σ, q', q'')
4                                   ⇒ ∃V. x ∈V ∧ (⟨q', V⟩ ∈eval(u) V
⟨q'', V⟩ ∈eval(w)) }⟩ | q ∈ Q }
5     s → Ξ := {⟨q, s⟩ | q ∈ I }
6   esac
7   for ⟨q, x⟩ ∈ Ξ
8     if (q ∈ Θ) Ξ := (Ξ ＼ {⟨q, V⟩}) ∪ {⟨q, V ∪ {v}⟩} fi
9   rof
10  return Ξ
11 corp
```

Figure 13

```
<html>
  <head><title>Sample</title></head>
  <body>
    <p>A paragraph</p>
    <div>
        <p>A paragraph in div</p>
    </div>
  </body>
</html>
```

Figure 14

eval("Sample") = ⟨ {0, 2}, "Sample" ⟩

Figure 15

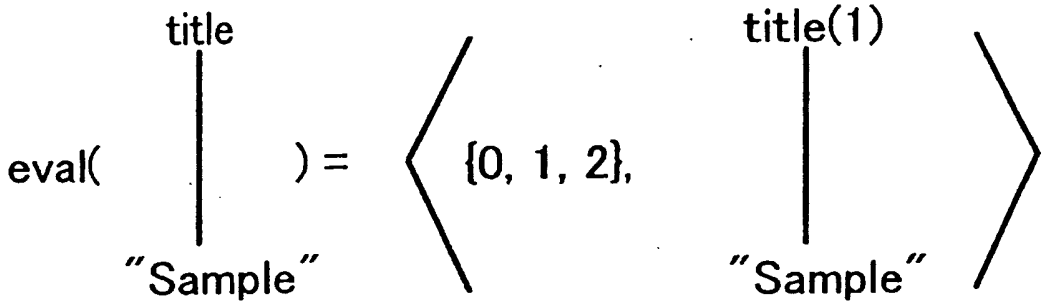


Figure 16

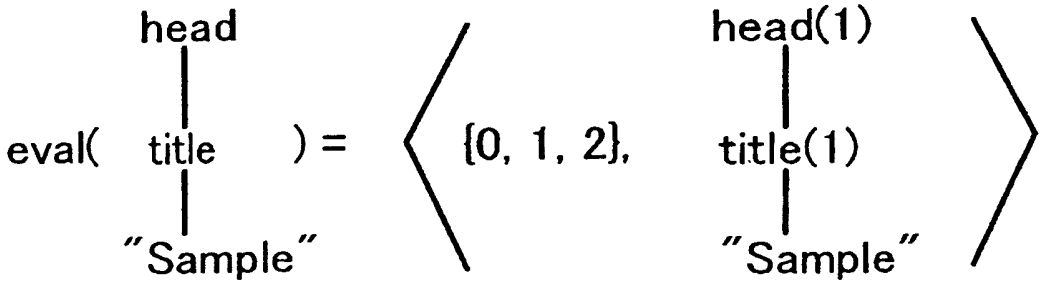


Figure 17

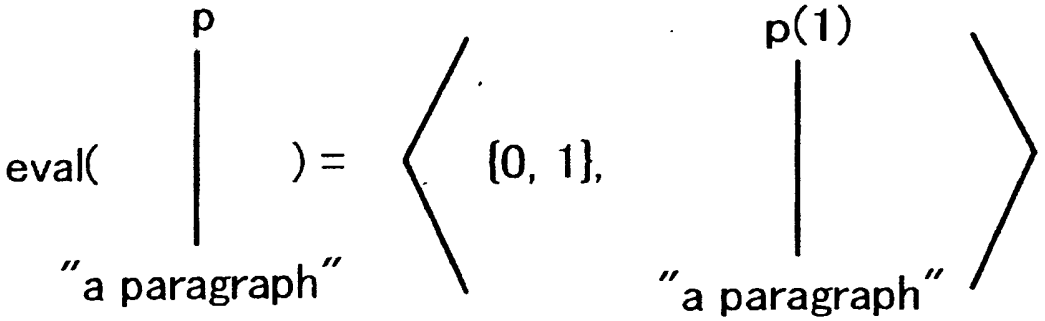


Figure 18

p
eval(
"a paragraph"
div
)=
p
"A paragraph in div"
{0, 1, 2},
p(0, 1)
"a paragraph"
div(1)
p(1)
"A paragraph in div"

Figure 19

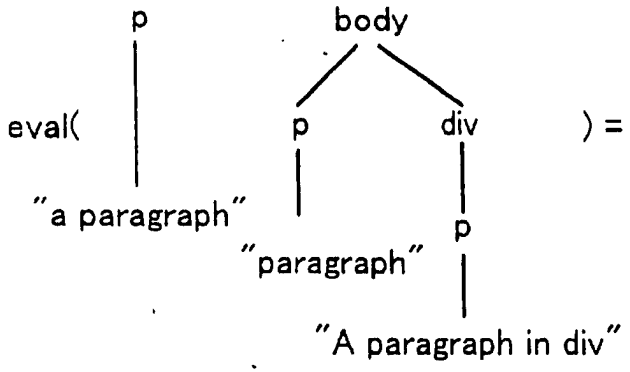
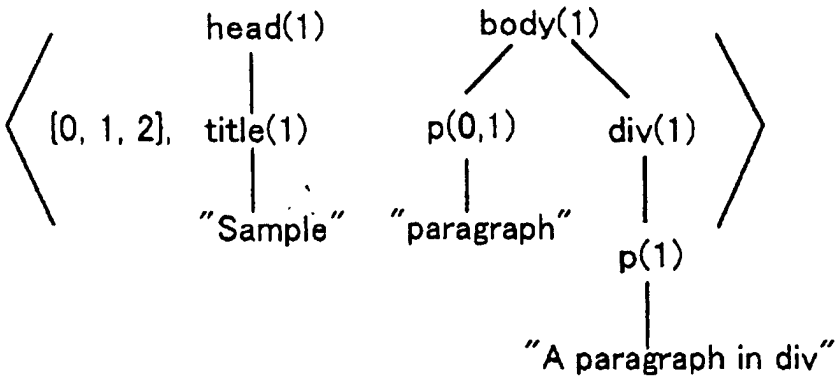

p
body
eval(
p
div
) =
"a paragraph"
"paragraph"
p
"A paragraph in div"

Figure 20


head(1)
body(1)
[0, 1, 2],
title(1)
p(0,1)
div(1)
"Sample"
"paragraph"
p(1)
"A paragraph in div"

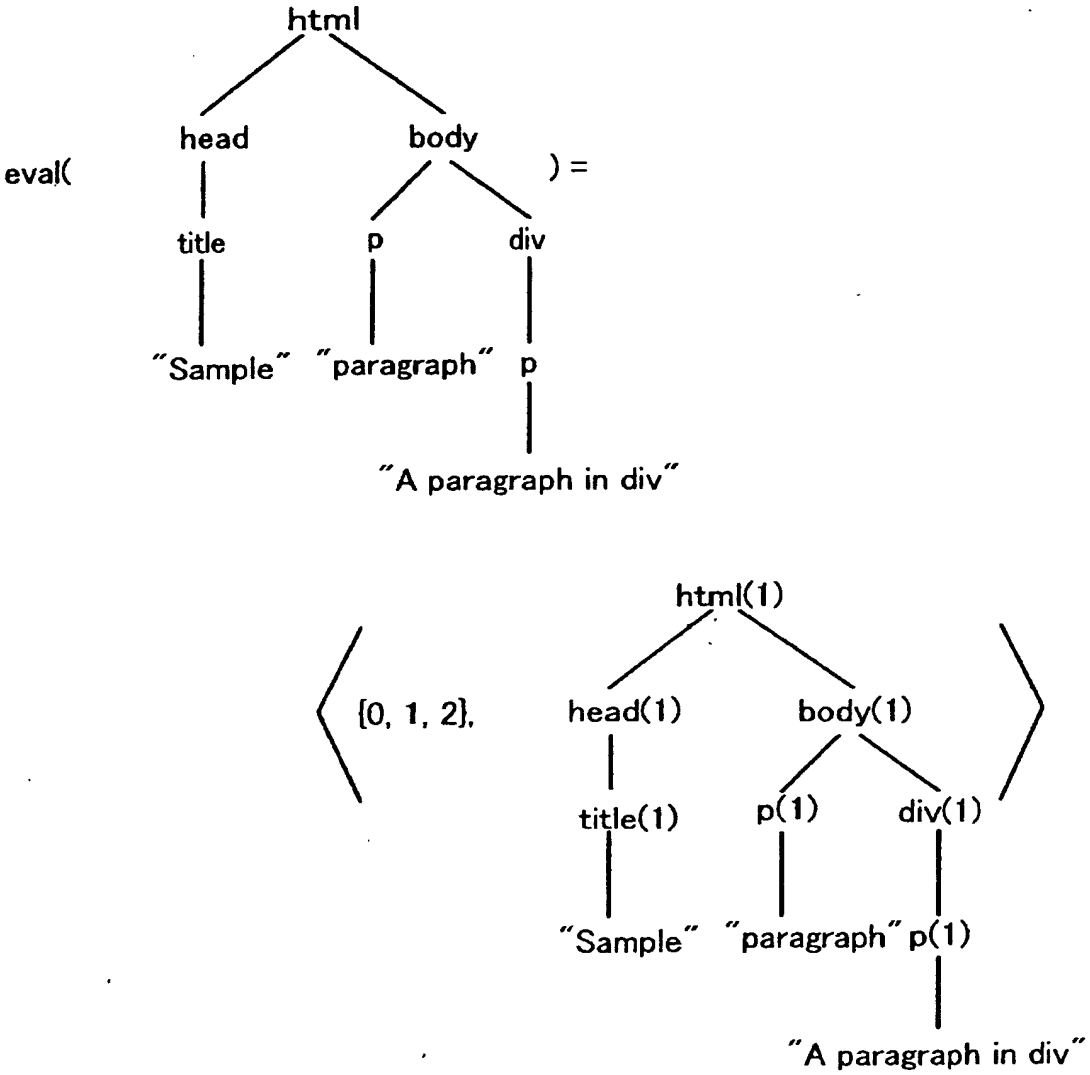

html
head
body
eval(
) =
title
p
div
"Sample"
"paragraph"
p
"A paragraph in div"
html(1)
{0, 1, 2},
head(1)
body(1)
title(1)
p(1)
div(1)
"Sample"
"paragraph"
p(1)
"A paragraph in div"

Figure 21

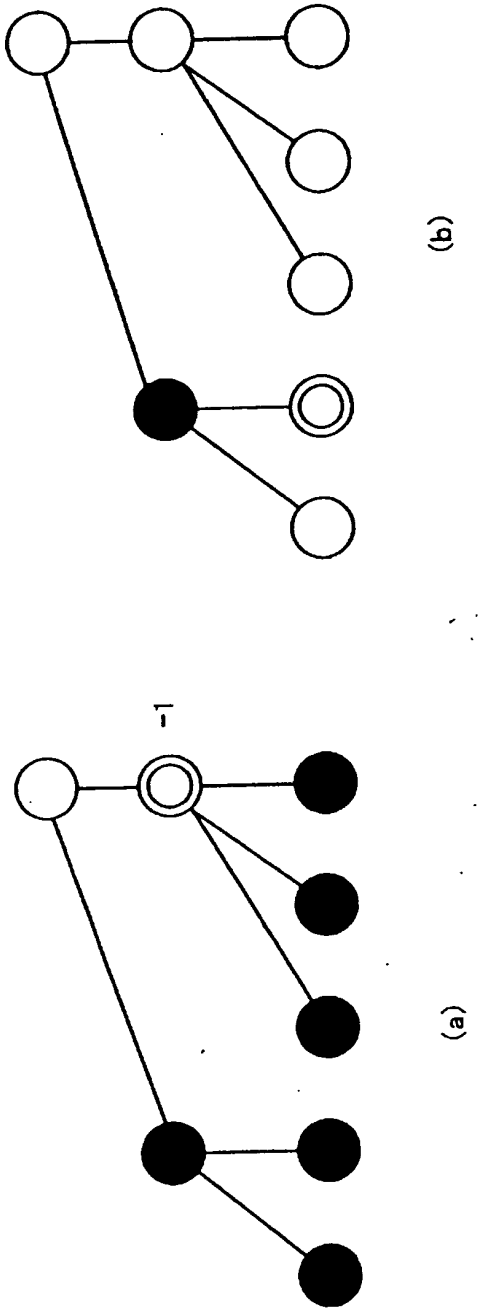


Figure 22

REACHED STATE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
$\neg a \wedge 1[1;2](\neg a \vee *) \wedge 2[1;2](\neg a \vee *)$	$\neg p$	$[1;2](\neg a \vee *)$	$[1;2](\neg a \vee *)$
$* \wedge 1[1;2](\neg a \vee *) \wedge 2[1;2](\neg a \vee *)$	any	$[1;2](\neg a \vee *)$	$[1;2](\neg a \vee *)$

Figure 23

REACHED STATE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	$\neg p$	0, 1	0, 1
1	any	0, 1	0, 1

Figure 24

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	[, F	$\neg p$	0, 1	0, 1
1	*, [, F	any	0, 1	0, 1

Figure 25

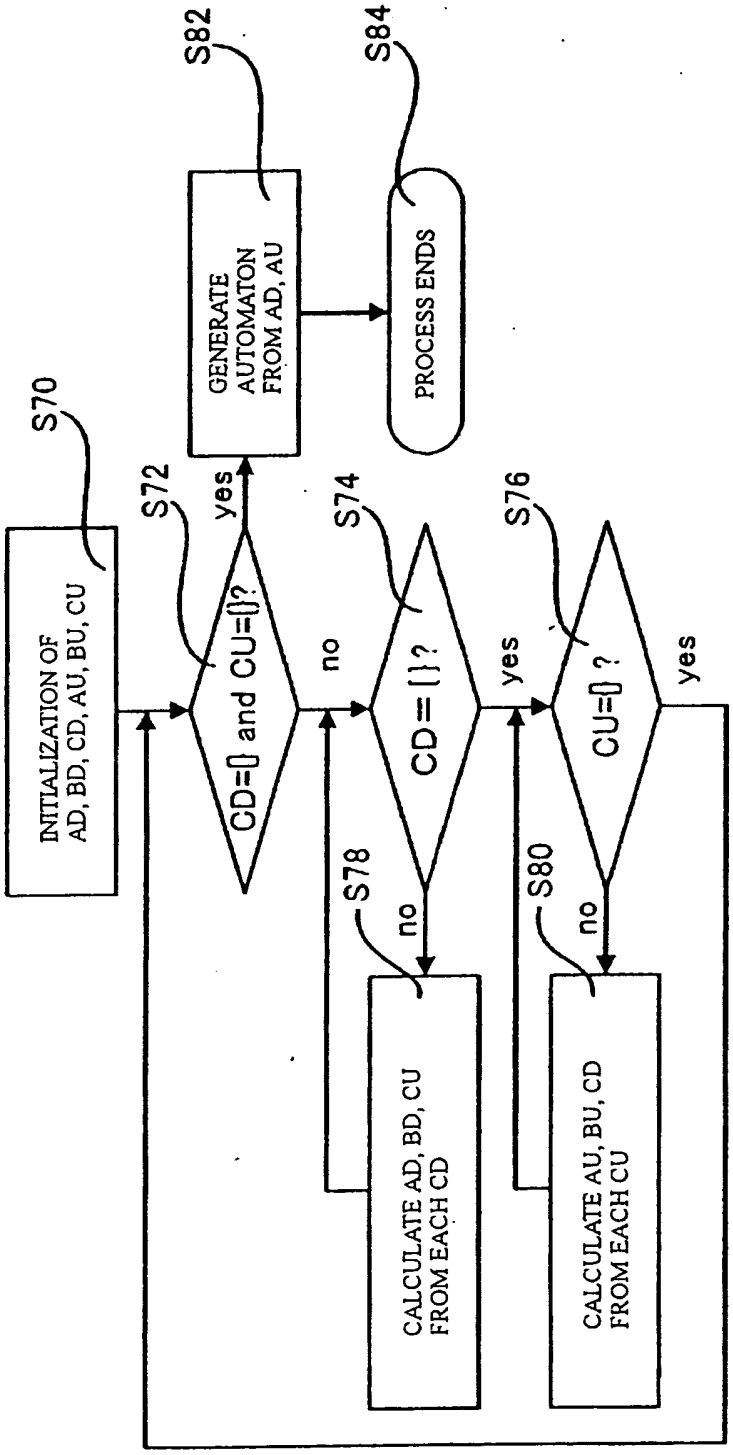

S70
INITIALIZATION OF AD, BD, CD, AU, BU, CU
S72
CD={} and CU={}?
yes
S82
GENERATE AUTOMATON FROM AD, AU
S84
PROCESS ENDS
no
S74
CD= {}?
yes
S78
no
CALCULATE AD, BD, CU FROM EACH CD
S76
CU={} ?
yes
S80
no
CALCULATE AU, BU, CD FROM EACH CU

Figure 26

```
1 proc main(φ)
2   Q := P := {T}, S := Resolved := {}
3   AD := BD := BU := CU := CD := {}
4   AU := {<T, T, T>, <T, T, ε>, <T, ε, T>, <T, ε, ε>}
5   T.succ1 := T.succ2 := T.prec1 := T.prec2 := ε.prec1 := ε.prec2 := {}
6   S-extend(T, {φ})
7   while CD ≠ {} or CU ≠ {}
8     for <q0, q1, q2> ∈ CD
9       CD := CD \ <q0, q1, q2>
10      X1 := S-extend(q1, q0.succ1)
11      X2 := S-extend(q2, q0.succ2)
12      for <_, q1', q2'> ∈ {q0} × X1 × X2 \ AD \ BD
13        if ¬∃ φ. -1φ ∈ (q1'.term \ q1.term) ∧ ¬∃ φ. -2φ ∈ (q2'.term \ q2.term)
14          then AD := AD ∪ {<q0, q1', q2'>}
15          else BD := BD ∪ {<q0, q1', q2'>}, CU := CU ∪ {<q0, q1', q2'>}
16        fi
17      CU := CU ∪ {q0} × (q1'.sub \ Resolved) ×(q2'.sub \ Resolved)
18        ∪ {q0} × {q1'} ×(q2'.sub \ Resolved) ∪ {q0} × (q1'.sub \ Resolved) × {q2'}
19      rof
20    rof
21    for <q0, q1, q2> ∈ CU
22      CU := CU \ <q0, q1, q2>
23      X := P-extend(q0, q1.prec1 ∪ q2.prec2)
24      for <q0', _, _> ∈ X × {q1} × {q2} \ AU \ BU
25        if ¬∃ φ. 1φ or 2φ ∈ (q0'.term \ q0.term)
26          then AU := AU ∪ <q0', q1, q2>
27          else BU := BU ∪ <q0', q1, q2>, CD = CD ∪ <q0', q1, q2>
28        fi
29        CD := CD ∪ (q0'.sub \ Resolved) × {q1} × {q2}
30      rof
31    rof
32  elihw
33  δ := {}
33  for <q0, q1, q2> ∈ AD ∪ AU
34    R0 := {q | q.base = q0 ∧ q ∈ Resolved ∩ S}
35    R1 := {q | q.base = q1 ∧ q ∈ Resolved ∩ P}
36    R2 := {q | q.base = q2 ∧ q ∈ Resolved ∩ P}
37    D := ({q0} ∪ R0) × ({q1} ∪ R1) × ({q2} ∪ R2)
38    for <q0', q1', q2'> ∈ D
39      δ := δ ∪ {(σ1∧...∧σn, q1', q2')→q0'} (σ1, ..., σn ∈ q0'.term)
40    rof
41  rof
42  F := {q | ∃ Φ. Φ ⊆ q.term ∧ Φ ∈ expand(φ)}
43  return <Q, F, δ>
44 corp
```

Figure 27

```
1 proc expand(Φ)
2   Ψ := {}
3  for φ ∈ Φ
4     Φ := Φ \ {φ}
5    case φ
6       σ → Ψ := Ψ ∪ {σ}
7       ψ ∧ ψ' → Φ := Φ ∪ {ψ, ψ'}
8       ψ ∨ ψ' → return expand(Φ ∪ Ψ ∪ {ψ}) ∪ expand(Φ ∪ Ψ ∪ {ψ'})
9       [m]ψ → Φ := Φ ∪ {ψ} ∪ {b[m]ψ | b ⊆ m}
10      <m>ψ → return expand(Φ ∪ Ψ ∪ {ψ}) ∪ ∪_{b ⊆ m} expand(Φ ∪ Ψ ∪ {b<m>ψ, (b)})
11      otherwise Ψ := Ψ ∪ {φ}
12    esac
13    fi
14  rof
15  return {Ψ}
16 corp
```

Figure 28

```
1 proc d-extend(q, Φ)
2   if q = ε return {ε}
3   X := {}
4   Ξ := expand(Φ)
5   if ∃ Ψ ∈ Ξ. Ψ ⊆ q.term then Ξ := {{}} fi
6   for Ψ ∈ Ξ
7     if ( q ∈ S ∧ d = S) ∨ ( q ∈ P ∧ d = P)
8       then q' := <q.term ∪ Ψ , q.base>
9       else q' := <q.term ∪ Ψ, q>
10    fi
11    X := X ∪ {q'}
12    if ¬(q' ∈ Q) then
13      Q := Q ∪ {q'}
14      q'.succ1 := {ψ | 1 ψ ∈ q'.term \ q'.base.term }
15      q'.succ2 := {ψ | 2 ψ ∈ q'.term \ q'.base.term }
16      q'.prec1 := {ψ | -1 ψ ∈ q'.term \ q'.base.term}
17      q'.prec2 := {ψ | -2 ψ ∈ q'.term \ q'.base.term}
18      if d = S then
19        S := S ∪ {q'}
20        if q'.succ1 = q'.succ2 = {} then Resolved := Resolved ∪ {q'}
21         else
22           q'.succ1 := q'.succ1 ∪ q'.base.succ1
23           q'.succ2 := q'.succ2 ∪ q'.base.succ2
24           CD := CD ∪ {<q', q1, q2> | <q'.base, q1, q2> ∈ AU ∪ BU, q1, q2 ∈ Q}
25          fi
26      else
27        P := P ∪ {q'}
28        if q'.prec1 = q'.prec2 = {} then Resolved := Resolved ∪ {q'}
29        else
30          q'.prec1 := q'.prec1 ∪ q'.base.prec1
31          q'.prec2 := q'.prec2 ∪ q'.base.prec2
32          CU := CU ∪ {<q0, q', q''> | <q0, q'.base, q''.base>
                ∈ AD ∪ BD, q0 ∈ Q, q'' ∈ P \ Resolved}
33              ∪ {<q0, q'', q'> | <q0, q''.base, q'.base>
                ∈ AD ∪ BD, q0 ∈ Q, q'' ∈ P \ Resolved}
34              ∪ {<q0, q', q2> | <q0, q'.base, q2> ∈ AD ∪ BD, q0, q2 ∈ Q}
35              ∪ {<q0, q1, q'> | <q0, q1, q'.base> ∈ AD ∪ BD, q0, q1 ∈ Q}
36        fi
37      fi
38    fi
39  rof
40  return X
41 corp
```

Figure 29